Towards Decentralised Workflow Scheduling via a Rule-driven Shared Space

Héctor Fernández[1], Marko Obrovac[2], and Cédric Tedeschi[2]
hector.fernandez@vu.nl, marko.obrovac@inria.fr, cedric.tedeschi@inria.fr

[1] VU University Amsterdam, The Netherlands
[2] IRISA. Université Rennes 1 / INRIA, France

Abstract. This paper addresses decentralised workflow scheduling, which calls for fulfilling two seemingly contradictory requirements: decentralisation and efficiency. We describe a two-layer architecture that allows decentralisation while making it possible for each scheduling decision to be taken based on a global perspective of the current state of resources. The first layer expresses the scheduling strategy on a global perspective, relying on a coordination space where workflows are first decomposed in tasks, and then tasks mapped onto resources. The second layer allows this global policy to be enacted in a fully-decentralised manner, based on a distributed hash table indexing resources, enhanced with advanced discovery mechanisms. Thus, in spite of decentralisation, the system is able to select the momentarily most appropriate resource for a given task, independently of the location of the provider of the resource. The framework's expectations in terms of scalability and network overhead are studied through simulation experiments.

1 Introduction

With the rise of Service Computing, a growing number of scientific applications are defined as *workflows of services*, *i.e.* temporal compositions thereof, which in turns intensifies the usage of distributed computing infrastructures, which consequently suffer from their centralisation, leading to reliability, privacy, and sustainability [1, 2]. These situations led to the advocation of decentralised way of building these infrastructures, based on the federation of distributed computing resources [2]. Unfortunately, decentralisation and efficiency may be contradictory objectives. As described in [2], some *coordination* between participants/nodes of the federation is required to provide some efficiency in spite of decentralisation. In this position paper, we focus on **workflow scheduling**, *i.e.*, the process deciding which resource each task of some workflow has to be run on, and how to provide this *coordination* specifically for scheduling. Coordination is needed in order to ensure the consistency of the scheduling decisions taken independently by distinct nodes scheduling different tasks.

Let us briefly review few recent approaches for fully decentralised schedulers. Works such as [3] motivate the need for interlinking computing platforms through peering arrangements enabling resource sharing. Local schedulers are connected

through *gateways* used to serve locally-unsatisfied requests. However, preferring locality might lead to an inefficient scheduling. Ranjan *et al.* [4] proposed a decentralised scheduler using a distributed hash table (DHT) split in regions through a P2P coordination space. The DHT acts as a distributed *blackboard* containing requests. Each region is managed by one peer, responsible for finding suitable resources in its part of the platform. Finally, works such as [5] propose gossip-based schemes to schedule computation-intensive jobs, where there are no predefined schedulers — any entity can schedule a job. However, the unstructured nature of gossiping protocols leads to only weak guarantees when searching a suitable resource for a task. The presented framework intends not only to decentralise the scheduling process, but also targets the possibility to support efficient scheduling algorithms through a coordination layer, built on top of the network, enabling a global knowledge of available resources.

Decentralisation and coordination cannot be tackled at once. We separate two concerns: *(i)* the specification of scheduling's logic; and *(ii)* its decentralised implementation. For the first problem, we need some high-level abstractions able to express these rules naturally. Rule-based programming, and in particular, the chemical programming model, offers adequate abstractions to specify coordination in distributed systems [6]. This model envisions a computation as a set of concurrent reactions between molecules of data. Formally speaking, the data is a multiset rewritten by a set of rules to be applied concurrently by distributed processes. Then, the second problem – the distributed implementation – can be refined as, how to distribute this multiset while being able to read and write it concurrently, so that the coordination is decentralised in its entirety. In this paper, we rely on HOCL (*Higher-Order Chemical Language*) [7], a rule-based language, enhanced with a chemistry-inspired execution model. According to the metaphor, molecules of data float in a solution, and, on collision, react according to reaction rules (the program) producing new molecules (the resulting data). In HOCL, the solution is a multiset containing molecules, and rewriting rules define reactions. The reactions take place in an implicitly parallel and non-deterministic way until no more reactions are possible — a stable state referred to as *inertia*. Let us consider the following chemical program which extracts the maximum value from a set of integers: **replace** x, y **by** x **if** $x \geq y$ **in** $\langle 2, 4, 5, 7, 9 \rangle$. The rule specifies that any pair of integers inside the solution can react, consuming these two molecules and creating a new one with the highest value of the two. The HOCL execution model simply assumes reactants are captured atomically, so each molecule reacts only once. The exact degree of parallelism and the order in which the rule is applied is left to the implementor of a runtime supporting the language. Thus, one of the possible execution is the following: $\langle 2, 4, 5, 7, 9 \rangle \rightarrow^* \langle 4, 5, 9 \rangle \rightarrow \langle 5, 9 \rangle \rightarrow \langle 9 \rangle$. In case new molecules (here, integers) are dynamically inserted into the multiset, an *imbalance* may arise, triggering new reactions. Enabling persistent coordination amongst scheduling entities requests for such a concept. We here use HOCL to express the scheduling process.

In the following, we present a two-layered, fully-decentralised workflow scheduling framework. The top layer is a *chemically*-coordinated shared space

where workflows are decomposed into tasks, which are mapped to resources. The bottom layer implements this shared space in a fully decentralised way, based on a peer-to-peer overlay network allowing the efficient storage and retrieval of molecules. The system proposed allows for a dynamic multiple-workflow scheduling. The conducted simulations allow to describe more precisely the scalability and overhead of such a platform. Section 2 describes our decentralised workflow scheduling system and its coordination model. Section 3 evaluates the performance and network overhead of the framework. Section 4 concludes.

2 A Distributed Shared Space for Workflow Scheduling

We now present a fully-decentralised, just-in-time, multiple-workflow scheduler, where the scheduling process is shared by a set of *chemical engines* running on every resource machine. As illustrated by Figure 1, the proposed system comprises a *communication* layer and a *coordination* layers.

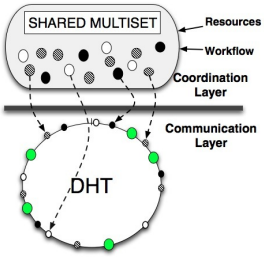

Fig. 1. Two-layer architecture.

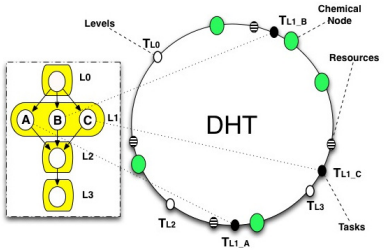

Fig. 2. Workflow decomposition.

Abstracting out the underlying network topology and dealing with the large number of resources, chemical engines, referred to as *nodes* in the remainder, are connected through a DHT [8], constituting the **communication layer**, and illustrated in the lower part of Figure 1. The DHT handles the dynamic nature of the platform while preserving a uniform and efficient communication pattern.

The DHT allowing chemical engines to share data (molecules) in a scalable fashion, a *distributed shared multiset* will be created on top of it, to which nodes expose their molecules representing workflow tasks and resources, as shown in the upper side of Figure 1. Thus, nodes can use molecules they do not hold, making it possible to build a **coordination layer** based on a *distributed scheduling space*. Each chemical engine is provided with rules to decompose the workflow and schedule the tasks, acting by consuming and producing molecules within the shared multiset.

2.1 Molecule Types

There are three types of molecules in our system: molecules representing workflow levels, task molecules and resource molecules, respectively. Each molecule

is assigned a unique identifier using the DHT's hash function. The molecule is then placed in the shared multiset (as depicted in Figure 2), by routing it to the appropriate node based on its identifier. Upon its entry in the system, a workflow is decomposed into levels by the entry node, producing **level molecules** (small white circles in Figure 2). Level molecules take the form LEVEL:$idLevel$:$\langle task_1, \ldots, task_n \rangle$, where $idLevel$ identifies this level in the workflow, and $task_1, \ldots, task_n$ are the tasks the level comprises. Once it is the turn of a level to be processed, the node storing its molecule splits it into a set of **task molecules** (black circles in Figure 2), one per task. A task molecule takes the form TASK:$idTask$:$\langle cmd$:$res\_desc \rangle$:$\langle$DEST:$destTaskId, \ldots \rangle$, where $idTask$ is the task's identifier, cmd denotes the actual service to invoke, res_desc is the description of the resource requirements, and the $\langle$DEST:$destTaskId, \ldots \rangle$ sub-solution specifies to which tasks the output of this task has to be sent. Physical resources are represented by **resource molecules** of the form RES:$idRes$:$\langle feature_1, \ldots, feature_n \rangle$, where $idRes$ is the identifier of the resource and $feature_1, \ldots, feature_n$ are its current characteristics, such as the number of processors, the CPU load, or the memory usage. Unlike level and task molecules, resource molecules are not uniformly hashed. Instead, they are kept on the originating node (as suggested in Figure 2). Doing so keeps the network cost of updating a resource molecule at zero.

2.2 Workflow Scheduling Process

The framework proposed aims at providing a decentralised *just-in-time* task-to-resource mapping, on top of which more complex workflow scheduling heuristics can be implemented. It follows a rule-based (event-driven) execution model, in which a rule is triggered when a node receives a molecule or a new workflow. Three entities can trigger a rule: a workflow, a level molecule and a task molecule. The chemical rules used for the scheduling process are given in Algorithm 1(down). A workflow initially enters the platform by being sent to a given node, its *entry point*, which receives the workflow and decomposes it in levels, producing level molecules. Workflows are described using the *chemical workflow definition* illustrated in Algorithm 1(up), where the main solution is composed of as many *task molecules* as there are tasks participating in the workflow.

Upon the receipt of a workflow, a node triggers the *workflowDecomp* rule which reorganises the tasks represented as sub-solutions of the workflow representation into levels. To distinguish between levels which can be scheduled and those which have to wait, we use two types of molecules: LEVEL:num:$READY$ and LEVEL:num. The initial workflow decomposition, through the activation of the rule *workflowDecomp*, produces only LEVEL:num molecules to indicate that none of the levels can be scheduled. However, as the scheduling goes on, these molecules will, one by one, turn into LEVEL:num:$READY$ molecules, indicating that the tasks of the previous levels have been completed and that the tasks of the next level can be scheduled for execution. To extract tasks from level molecules, a node uses the *levelDecomp* rule. This rule consumes a level molecule with its state set to $READY$, and produces as many task molecules

Algorithm 1 Chemical workflow representation (up); and Generic rules (down).

```
1.01 ⟨ TASK : 1 : ⟨cmd_1 : res_desc_1⟩ : ⟨DEST : 2, DEST : 3, . . . ⟩,
1.02   TASK : 2 : ⟨cmd_2 : res_desc_2⟩ : ⟨DEST : 4, . . . ⟩,
1.03   TASK : 3 : ⟨cmd_3 : res_desc_3⟩ : ⟨DEST : 4, . . . ⟩,
1.04   TASK : 4 : ⟨cmd_4 : res_desc_4⟩ : ⟨⟩    ⟩
2.01 let workflowDecomp = replace ⟨ TASK_1, ..., TASK_n ⟩
2.02          by LEVEL:1:⟨ TASK_1 ⟩, ..., LEVEL:L:⟨ TASK_n ⟩
2.03 let levelDecomp = replace-one LEVEL:num:READY:⟨ TASK_1, ... ,TASK_n ⟩
2.04          by TASK_1, ... ,TASK_n
2.05 let mapTaskRes = replace TASK_i, RES_j  by system.deploy(TASK_i, RES_j )
2.06          if (TASK_i.isCompatibleWith(RES_j))
```

as there are tasks in the given level (Algorithm 1, line 2.03). Upon the receipt of a task molecule, a node uses the *mapTaskRes* rule to map the task to the best resource it can find (Algorithm 1, line 2.05). For this purpose, we use a second, order-preserving DHT layer, physically matching the first one, to store *meta-molecules — pointers* to resource molecules. A meta-molecule is placed in this layer according to its value, *i.e.* cpu usage. When a node looks for a resource to execute its task on, it sends a request in this second layer. Using the resources' requirements indicated in the task molecule, the second DHT layer is scanned by a range query (whose complexity is typically in $O(log^2(n))$) [9], such as $cpu < 80\%$, reflecting the **if**-clause of the *mapTaskRes* rule. If a matching resource molecule is found, the rule produces a molecule that deploys the given task onto the resource thus found (denoted by the *system.deploy()* molecule in Algorithm 1). When all of its tasks have been completed, the node holding the (currently active) level molecule retrieves the inactive molecule of the next level, to allow the next level to be processed. Once the tasks of the last level have completed, its responsible node collects all of the results and transfers them to the entry node, which delivers them to the client that submitted the workflow for execution. Note that, due to decentralisation, multiple workflows can be scheduled at the same time. They can be processed in parallel, each being managed by a different set of nodes, while the final scheduling decisions are still taken on a global perspective. They are independently decomposed, each on a different entry-point node, but their tasks compete collectively for resources.

3 Preliminary Evaluation

We built a discrete-time simulator in Python to simulate the framework when multiple randomly-generated workflows are executed. As shown by Figure 3, increasing the number of nodes has an impact on the time taken to schedule workflows which is limited, as the routing's cost grows logarithmically with the number of nodes. Also, thanks to decentralisation, increasing the number of workflows does not augment much the time to solve them, as it enables a high degree of parallelism. Figure 4 shows that the logarithmic nature of the routing

limits the impact of increasing the number of nodes on the network congestion. The number of messages naturally increases proportionally with the number of workflows, as the scheduling of a task relies on resource retrieval, which needs $O(log^2(n))$ messages to complete. Finally, Figure 5 suggests that the number of messages sent per node drastically reduces when more nodes take part in the scheduling process, even when an elevated number of workflows is present.

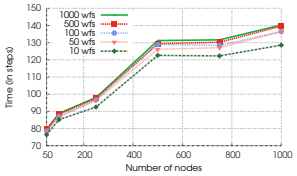

Fig. 3. Execution time.

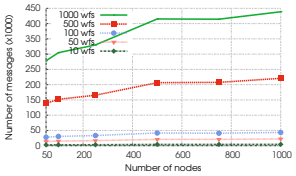

Fig. 4. Network traffic.

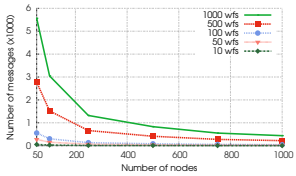

Fig. 5. Traffic per node.

4 Conclusion

To address the contradictory objectives of decentralisation and efficiency of scheduling, we have proposed a high-level coordination mechanism relying on a chemistry-inspired, rule-based programming model, and supported by a DHT-based decentralised architecture to retrieve and match tasks and resources efficiently. This *DHT-driven* coordination enables *just-in-time* scheduling, allowing to match each task to the momentarily best resource available. Simulations were conducted, showing further the feasibility and scalability of the approach.

References

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," tech. rep., UC Berkeley, 2009.
2. A. Marinos and G. Briscoe, "Community cloud computing," in *CloudCom*, 2009.
3. K. Leal, E. Huedo, and I. M. Llorente, "A decentralized model for scheduling independent tasks in federated grids," *FGCS*, vol. 25, 2009.
4. R. Ranjan, M. Rahman, and R. Buyya, "A decentralized and cooperative workflow scheduling algorithm," in *CCGRID*, IEEE, 2008.
5. X. Vasilakos, J. Sacha, and G. Pierre, "Decentralized As-Soon-As-Possible grid scheduling: A feasibility study," in *ICCCN*, 2010.
6. H. Fernández, C. Tedeschi, and T. Priol, "A Chemistry-Inspired Workflow Management System for Scientific Applications in Clouds," in *e-Science*, 2011.
7. J. Banâtre, P. Fradet, and Y. Radenac, "Generalised multisets for chemical programming," *Mathematical Structures in Computer Science*, vol. 16, no. 4, 2006.
8. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *LNCS*, vol. 2218, 2001.
9. L. Chen, K. Candan, J. Tatemura, D. Agrawal, and D. Cavendish, "On overlay schemes to support point-in-range queries for scalable grid resource discovery," in *International Conference on Peer-to-Peer Coomputing*, 2005.